Hierarchy, Labels and Motion Description

GEOFF WYVILL, ANN WITBROCK, and ANDREW TROTMAN

Abstract

When we use a hierarchical model, we have to provide a means for the animator to refer to the individual parts. A labelling system is described for doing this. The idea is applied in an experimental system for controlling a simple human model performing acrobatics.

key words: Computer animation, hierarchy, scripts, kinematics

1. Introduction and context

In this paper we examine the labelling of parts in a hierarchy in the context of *scripted* animation of synthetic actors in three dimensions.

In traditional animation, every aspect of every frame is individually controlled. An automated animation system aims to provide the artist with detailed control at less cost. There have been many ways proposed to do this, ranging from a crude inbetweening of keyframes via parametric interpolation to rule-based kinematic algorithms and dynamic systems using physical laws. In describing these techniques, the emphasis has, quite reasonably, been on how each motion is described. In this paper, we address a related problem: how should motion description be related to the parts of a complicated model?

For the purpose of this discussion, we assume that both the modelling and animation are done by means of a *script*. That is, a set of instructions that can be laid out in text form. This is not meant to suggest that we recommend animating this way. When the basic tools for describing motion of characters have been established, the script elements may well be entered using an interactive interface. The use of a script is a device for discussing these elements.

Ideally, we should give directions to a synthetic actor as we would to a human actor. For example: "Walk to the desk and pick up the telephone." Of course, instructions at this level, leave out a great deal of detail. We expect the human actor to use a lot of knowledge: to recognise the desk with the telephone on it; to avoid other actors on the stage; to pick up the receiver, not the whole telephone; and probably, to hold the appropriate part of the instrument to her ear.

N. M. Thalmann et al. (eds.), *Creating and Animating the Virtual World*

At the low-level of our scripts, we are more concerned with such problems as making sure that all the arms and legs move with the actor and that the hand that picks up the telephone arrives in the right place and is still connected to the actor's arm when the contact is made.

All the experiments in this research have been conducted using the *Katachi* system at the University of Otago. *Katachi* is an experimental system for modelling, animation and rendering. It has no user interface and everything is done by scripts represented by "C" programs, or data files. *Katachi* uses CSG models and everything in a scene is represented by a hierarchy of components. Thus our concern is how to access and control this hierarchy in a way that an animator can understand.

2. Background

We make no attempt to summarize the mass of literature on human animation. Norman Badler (1986) has done that very well. Here we mention only work that is relevant to the question: "What can an animation script include?"

Dynamics can generate motion that is guaranteed to be natural because it follows physical laws. The work of Baraff (1990) demonstrates this with colliding falling objects. But for human animation, we do not know what forces are being applied to all the joints. Wilhelms and Barsky (1985) give a detailed formalisation to the problem of applying dynamics to an articulated body and Wilhelms (1986) discusses a practical way to specify the motion kinematically while using the results of dynamics. Various forms of an approximate and/or hybrid approach have been suggested, see Armstrong (1985, 1986), Wilhelms (1988), Isaacs (1988), Boulic (1990) and van Overveld (1990, 1991).

The idea of using a script for building objects comes from PDL-2 (Wyvill 1975). PDL-2 is a language for defining 2D pictures composed of lines only but it features labels. ASAS (Reynolds 1982) uses scripts with a LISP syntax, and describes 3D models and their animation. Maiocchi and Pernici (1990) describe how to use a library of recorded human motion to generate natural movement in artificial figures. Their approach is object oriented and uses the script concept. Similarly CINEMIRA (Thalmann 1990) animates from a script. In its simplest form, a script is kinematic in nature, but it doesn't have to be. David Zeltzer (1982) used functional control elements in the form of local programs to control parts of a motion. Ideally, we would like our script to operate at every level from a truly goal directed instruction (Drewery 1986) through remembered actions, dynamics or procedures to create default or usual motion, to detailed kinematics where the user is given absolute control, but must pay for it by having to keep track of all the details.

3. Problems inherent in hierarchy

The advantages of hierarchy in modelling are so obvious that we are rarely directed to the problems. A simple, humanoid actor is described as a tree structure of components as suggested in Fig. 1. Within the computer system, each component can be represented

by a record containing pointers to its sub-components. With each pointer is associated information telling us where the sub-component is and in what rotation. This information is conveniently encoded in transformation matrices and by traversing the tree, we can generate the absolute position of each part, by systematically concatenating the transformations of its ancestors in the structure.

This means that the animator can move the actor knowing that all the parts will move together. She knows that the arms will move with the body and the hands will move with the arms. She knows where the hands are: at the ends of the arms. She knows where the arms are: attached to the shoulders. There is no need to know where the hands are in global space, until the actor is required to pick something up: with a hand.

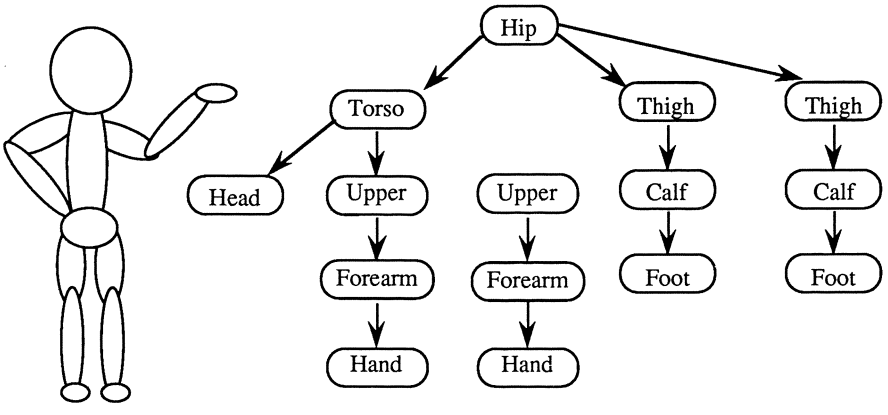


Fig. 1. Eve: a simple humanoid figure and tree structure.

At this point, we need to know where the hand is. More than that, we need to make sure that the whole body motion has not made it impossible for the hand to reach its target position. This defines our first problem: how does the animator identify, position and orient a given component in the hierarchy?

Suppose our first actor, Eve, presents the second, Adam, with an apple. To ensure that the apple moves with and remains in Eve's hand, it is attached to the hierarchy that defines Eve. At some point, it has to be transferred to the hierarchy that defines Adam. How is this to be specified?

The third problem concerns the control of a virtual camera in the scene. We would like to display Eve's view of Adam as he walks towards her. The camera is located at Eve's head and is to be kept pointing at Adam as they both move.

4. Labels in the hierarchy

Labelling the nodes of a hierarchy is not a new idea. Almost every computer file directory does exactly that. Labels were used in the PDL-2 language (Wyvill 1975) much as they are in the Katachi system for building models except that PDL-2 was strictly 2D and was not used for animation.

Objects in Katachi are represented by nodes that have two kinds of pointer, horizontal and vertical. A collection of nodes linked by horizontal pointers represent immediate components of the same object. A vertical pointer defines the object which is the component, Fig. 2.

Each node contains transformation matrices to say where the component is in the current object. Thus, in Fig. 2, the transformation of "right_arm" also applies to "upper", "forearm" and "hand". A label is a text field that may be added to any node. It labels that node and its associated local coordinate system. Thus the label "right_arm" in Fig. 2, labels the coordinate system into which "upper", "forearm" and "hand" are transformed.

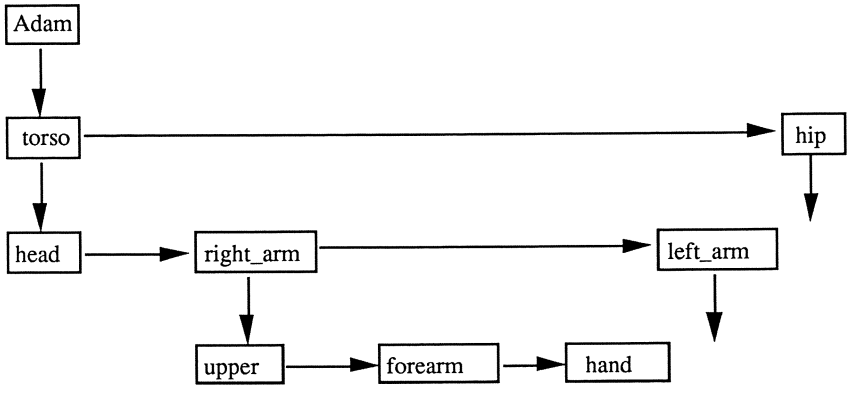


Fig. 2. Katachi structure.

The basic operation on labels is to identify the position of a label within an object. This 'position' is actually represented by a pair of transformation matrices, fore and aft, that represent instructions to transform a point from or to the coordinate system of the labelled component. We represent this by the function find:

```
int find(label, ob, fore, aft)
char *label;
object ob;
matrix fore, aft;
```

Find returns as its value, the number of occurrences of the label. The matrices fore and aft have their values set to represent the first occurrence.

Duplication of labels is permitted because parts in the hierarchy are duplicated. It would be unnatural to prevent the left hand from having a label "index_finger" because that label had already been used in the right hand. When we find a label, we avoid ambiguity by specifying an additional label as an ancestor of the target, e.g:

```
find( "right_arm/index_finger", Adam, fore, aft)
```

first finds the label "right_arm" and then searches the sub-structure for the label "index_finger". Notice that it is not necessary to describe the full path name which could be "right_arm/hand/index_finger", it is sufficient to specify the desired label and enough ancestors to avoid ambiguity.

5. Shared components

In engineering design applications, it is reasonable for the hierarchy to include shared components. This reduces the size of the data structure and maintains the principle that one data item is stored in only one place. If we are representing the design of a car, it is reasonable to keep only one copy of the wheel design and refer to it four times in the structure. Should we modify the wheel, all four wheels will change together.

For animation, this is no longer appropriate. The right and left hands may share a basic structure, but they will have different shapes at different times. One hand may be open when the other is closed. In a purely constructive system, we would build two hands with different structure to reflect their different shapes. This implies rebuilding the hierarchy for each frame of the animation. We prefer to regard the hierarchy as something that can be modified. So we build a right arm and duplicate it to make the left arm. (The positioning of the left arm will include a reflection.) Thus in pseudocode we may build:

```
hand = palm(...)        + digit("index_finger", where1)
                        + digit("middle_finger", where2)
                        + digit("ring_finger", where3)
                        + digit("little_finger", where4)
                        + digit("thumb", where5);
forearm = limb(...)     + transform(hand, where6);
right_arm = limb(...) + transform(forearm, where7);
left_arm = transform(duplicate(right_arm), where8);
```

Palm, digit and limb are routines that build those parts. Digit, by implication, has two arguments: the label and some position information encapsulated in the arguments where1, where2 etc.

6. Nested coordinate systems

Every object in the hierarchy has its own coordinate system. The coordinate system of a whole scene corresponds to the root of the hierarchy and is sometimes called "world space", but it is no different from any of the other systems. If our camera is positioned within a component, then our viewing system is, conceptually, in the coordinate system of that component.

A pointer into the hierarchy gives us direct access to the definition of a component in its own space. Thus in the example above, the variable hand points directly to the right hand.

find("index_finger", hand, fore, aft)

gives us the position of "index_finger" in the local space of hand whereas

find("index_finger", left_arm, fore, aft)

finds the "index_finger" of the left hand in the coordinate system of the left arm.

Having simultaneous access to these spaces means that we can change the position of the finger in the hand-space to close the hand and instantly find the position and rotation of the finger in the arm-space.

7. Application of labels

The labels immediately offer solutions to the problems of Section 3. The first problem, "where is Eve's right hand?" becomes trivial:

find("right_arm/hand", Eve, fore, aft);

tells us all we need to know.

In the second problem, Eve gives an apple to Adam; to which hierarchy is it attached? We don't attach the apple to Eve's hand directly. The apple belongs to the scene rather than to the component that is Eve.

```
scene = Eve + Adam + Apple;
find("Eve/right_arm/hand", scene, apple.fore, apple.aft);
```

gives the apple the same transformations as Eve's right hand. Later, we can use:

find("Adam/right_arm/hand", scene, apple.fore, apple.aft);

to put it into Adam's hand. Of course, in practice, the apple will not be placed where the hand is, but close to it, so this is a simplification.

The third problem was to place a camera for tracking. A camera has a position, a view angle, a point of attention and an 'up vector'. Each of these attributes can be set separately using the labels. We place the camera using:

find("head", Eve, fore, aft)

extracting the position data alone from fore and aft. The point of attention is gained from:

find("head", Adam, fore, aft)

and the 'up vector' can be fixed in relation to the scene. In our present system, we have to 'find' the camera for each frame. It might be better to define find as a function attached to the hierarchy so the camera move would follow the motion of Adam and Eve automatically, but, at present, our hierarchy does not include functions.

8. Building a body

It is not difficult to construct an articulated body with the CSG hierarchy alone but for the animation script we need more information about each body-part. This could be built into the CSG hierarchy but that would turn it into a specialised system for articulated bodies. Instead, the additional information is kept in an auxiliary structure. The basic unit is a list structure that mirrors the CSG structure. Horizontal pointers link items to a common root and vertical pointers refer to sub-groups.

Each node of this body-structure contains the following additional information:

- a path name to link to the actual part in the CSG tree,
- a path name to say which part it is linked to in the body,
- a mass and centre of gravity -- (Other information for dynamic calculations could be added here.)
- the position of the point of attachment in local coordinates,
- the position of the place in the parent component where it is attached,
- a list of rotation instructions for the animation.

This information can be created algorithmically or read from a file. Either way it is effectively constructed from a script after the manner of Reynolds (1982). The following is a fragment of the file for creating a person:

chest	partname
torso	part attached to
solid_sphere	rigid part description
0.7, 0.4, 0.7	scale(X,Y,Z)
brown plain	colour texture (repeated values for complex objects)
0.0, 0.0, 1.0	centre to joint
0.0, 0.0, 0.6	joint to parent's joint
0.13	proportion of mass
YZ	default plane of rotation
n	any initial rotation?
...	various rotation specifications

This creates a part called chest connected to torso. Each of these parts is made from a solid sphere stretched into an ellipsoid. Then follow colour and texture information and the attachment coordinates.

An important feature of this approach is that the components can be specified in any order because the linkage is given explicitly.

The spheres are already available as CSG primitives. If a more complicated, rigid shape is required, it can be built using the underlying CSG system, given a name and then asked for in the script just like the primitive shapes. Mary-Lou's head (Figs. 3 and 4) was constructed like that.

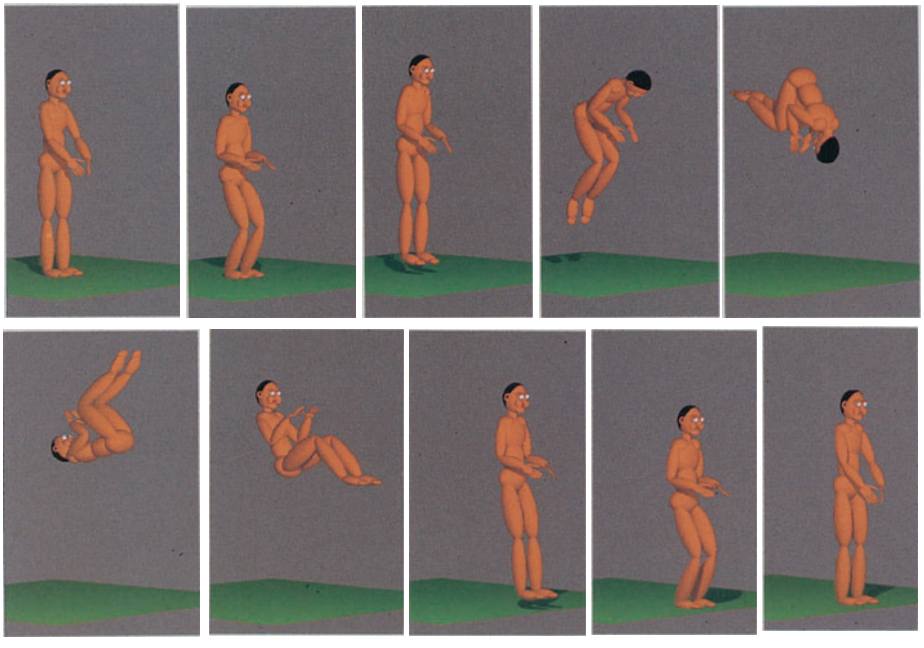


Fig. 3. Frames showing Mary-Lou performing a somersault.

9. Motion scripts

The whole idea of this structure is that we can mix dynamics, constraints and direct kinematic instructions to control our figures. The simplest example is fully kinematic. We provide lists of rotations for components to take place during specific time intervals. The following file fragment describes the action of Mary-Lou's left leg during the somersault, Fig 3.

```
lfoot
0.0,0.25 ZY 30.0 0.5 0.5
lfoot
0.25,0.3 YZ 30.0 0.5 0.5
lcalf
0.0,0.25 YZ 60.0 0.5 0.5
lcalf
0.25,0.3 ZY 60.0 0.5 0.5
lthigh
0.0,0.25 ZY 30.0 0.5 0.5
```

The first entry reads: during the time 0.0 to 0.25 rotate the left foot (lfoot) from Z to Y in local coordinates, by 30 degrees with uniform acceleration for half the time and uniform deceleration for the second half.

But the somersault motion is not quite that simple. During the first part of the motion, Mary-Lou's toes are bound to the ground plane. This means that all the angular changes cause the body to drop and rise again. At a pre-determined time, the toes are released and Mary-Lou's body is free to rise.

At this point, it would be appropriate to switch to a dynamic model. In fact we continue with a script that takes the whole body up and down with uniform acceleration and spins it with uniform accleration/deceleration. As Mary-Lou's toes contact the ground, we switch back to control her from joint angles again.

10. Reaching

A feature of the structure with labels is that there is no implication that a body is placed where the root node is. We can find such things as the centre of gravity or the position of the left toe and fix the relative motion from there. We would like to be able to ask the question "Can Mary-Lou reach the ball from where she is standing?" This implies two fixed points and asks us to deduce something about the linkage of the body to be able to reach both. We have implemented a very simple algorithm to do this based on the idea of the effective length of a chain of joints. In effect, we first ask if Mary-Lou can touch the ball with her hand. This she can do if the wrist is near enough. Otherwise, we look at the hand plus forearm and see if the pair of joints can make the right distance from the shoulder. If not, we look at other joints to see if the shoulder can be made nearer, and so on.

Figure 4 shows an example of this algorithm working as expected. Mary-Lou extends her arm and touches the ball. Figure 5 shows the algorithm working correctly, but not as expected. The ball is placed near the viewer, half an arm's length from Mary-Lou. Since she cannot reach the ball with her hand, Mary-Lou moves her forearm, to put the hand nearer the ball. She still needs more distance and gets it by extending from the shoulder and bending forward. Although she can reach the ball by this method, a more natural solution would be to twist the body and lean forward or to turn and take a step towards the ball first.

11. Conclusion

We have implemented a simple hierarchy to represent geometric models and provide a system of labels on selected components. Use of the labels solves some of the classical problems of hierarchy and gives us a base on which to build animation rules. So far, we have kept to simple scripts augmented by a minimal reaching algorithm. The structure gives us an environment for the implementation of more complicated algorithms and constraints.

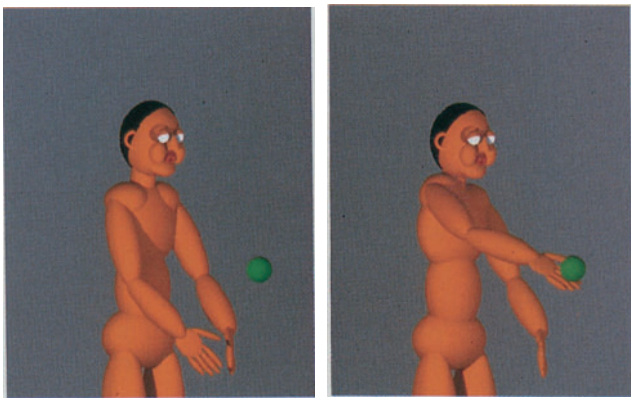


Fig. 4. Mary-Lou performs a simple reach.

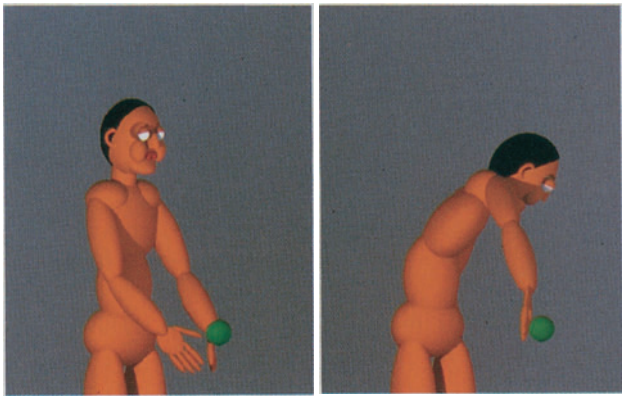


Fig. 5. Mary-Lou contorts herself in an effective if unusual reach.

12. References

Armstrong WW, Green M, Lake R(1986) Near-Real-Time Control of Human Figure Models, Proceedings of Graphics Interface '86:147-151

Armstrong WW, Green MW (1985) The dynamics of articulated rigid bodies for purposes of animation, *The Visual Computer* 1(4): 231-240

Badler NI (1986) Animating Human Figures: Perspectives and Directions, Proceedings of Graphics Interface '86: 115-120

Baraff D (1990) Curved surfaces and coherence for non-penetrating rigid body simulation, *Computer Graphics,* SIGGRAPH '90 Proceedings 24(4): 19-28

Barzel R, Barr AH (1988) A Modeling System Based on Dynamic Constraints, *Computer Graphics,* SIGGRAPH '88 Proceedings, 22(4): 179-188

Boisvert D, Magnenat-Thalmann N, Thalmann D (1989) An Integrated Control View of Synthetic Actors, *New Advances in Computer Graphics,* Proc. CG International '89, Springer, 277-288

Boulic R, Magnenat Thalmann N, Thalmann D (1990) A global human walking model with real-time kinematic personification *The Visual Computer* 6(6): 344-358

Bruderlin A, Calvert TW (1989) Goal-Directed, Dynamic Animation of Human Walking *Computer Graphics,* SIGGRAPH '89 Proceedings, 23,(3): 233-242.

Drewery K, Tsotsos J (1986) Goal Directed Animation using English Motion Commands, Proceedings of Graphics Interface '86: 131-135.

Getto P, Breen D (1990) An object-oriented architecture for a computer animation system, *The Visual Computer,* 6(2): 79-92

Isaacs P M, Cohen M F (1988) Mixed methods for complex kinematic constraints in dynamic figure animation, *The Visual Computer,* 4(6): 296-305

Maiocchi R, Pernici B (1990) Directing an animated scene with autonomous actors, *The Visual Computer,* 6(6): 359-371

Reynolds C (1988) Computer Animation with Scripts and Actors *Computer Graphics,* SIGGRAPH '82 Proceedings, 16(3): 289-296.

Magnenat Thalmann N, Thalmann D (1990) *Computer Animation Theory and Practice,* Second Edition Springer-Verlag

van Overveld CWAM (1990) A Technique for Motion Specification in Computer Animation, *The Visual Computer,* (6): 106-116

van Overveld CWAM (1991) An iterative approach to dynamic simulation of 3-d rigid-body motions for real-time interactive computer animation, *The Visual Computer,* 7(1): 29-38.

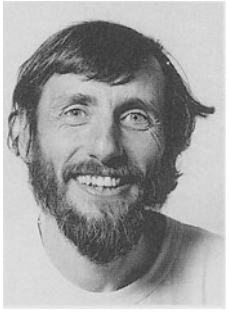
Wilhelms J P, Barsky B A (1985) Using dynamic analysis to animate articulated bodies such as humans and robots, *Computer-generated images: The state of the art,* Proceedings of Graphics Interface '85: 209-229

Wilhelms J (1986) VIRYA - A Motion Control Editor for Kinematic and Dynamic Animation, Proceedings of Graphics Interface '86:141-146.

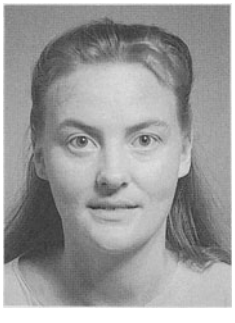
Wilhelms J, Moore M, Skinner R (1988) Dynamic animation: interaction and control, *The Visual Computer,* 4(6): 283-295

Geoff Wyvill (1975) Pictorial Description Language, *Interactive Systems,* Proceedings of Eurocomp: 511-526

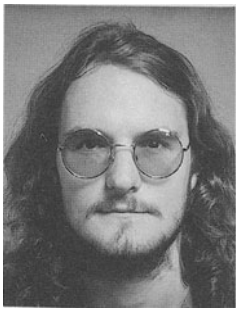
Zeltzer D (1982) Motor Control Techniques for Figure Animation, *IEEE CG&A* 2(9): 53-59



Geoff Wyvill received a BA from Oxford University, MSc and PhD degrees from the University of Bradford. He worked as a research physicist for the British Petroleum Company and lectured in computer science at the University of Bradford from 1969 to 1979. Since then he has been at the University of Otago where he runs the computer graphics laboratory. He is currently on sabbatical leave at EPFL, Lausanne, Switzerland. His main research interests are in geometric modelling and animation. He is best known for his work in CSG and implicit surface modelling. He is on the editorial boards of The Visual Computer and Visualization and Computer Animation and a member of SIGGRAPH, ACM, CGS and NZCS.



Ann Witbrock is a graduate student at the University of Otago. Before entering Computer Science, Ann trained as a dancer for 12 years. She completed her NZCE in Electronics and Computer Technology at Christchurch Polytechnic in 1986 and her BSc at the University of Canterbury in 1989. Her current research interests include Hierarchical Animation, Motion Definition, Dance, and CSG.



Andrew Trotman is a graduate student at the University of Otago. He completed his BA in Computer Science during 1988. Andrew's research interests include CSG, Ray Tracing, Implicit Surfaces, algorithmic efficiency and algorithmic correctness. Andrew is a member of SIGGRAPH and ACM.